Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

Applications of Lexicographic Semirings to Problems in Speech and Language Processing†

Richard Sproat*
Google, Inc.

Mahsa Yarmohammadi**
Oregon Health & Science University

Izhak Shafran**
Oregon Health & Science University

Brian Roark*
Google, Inc.

This paper explores lexicographic semirings *and their application to problems in speech and language processing. Specifically, we present two instantiations of binary lexicographic semirings, one involving a pair of tropical weights, and the other a tropical weight paired with a novel string semiring we term the* categorial semiring. *The first of these is used to yield an exact encoding of backoff models with epsilon transitions. This* lexicographic language model semiring *allows for off-line optimization of exact models represented as large weighted finite-state transducers in contrast to implicit (on-line) failure transition representations. We present empirical results demonstrating that, even in simple intersection scenarios amenable to the use of failure transitions, the use of the more powerful lexicographic semiring is competitive in terms of time of intersection. The second of these lexicographic semirings is applied to the problem of extracting, from a lattice of word sequences tagged for part of speech, only the single best-scoring part of speech tagging for each word sequence. We do this by incorporating the tags as a categorial weight in the second component of a ⟨Tropical, Categorial⟩ lexicographic semiring, determinizing the resulting word lattice acceptor in that semiring, and then mapping the tags back as output labels of the word lattice transducer. We compare our approach to a competing method due to Povey et al. (2012).*

1. Introduction

Applications of finite-state methods to problems in speech and language processing have grown significantly over the last decade and a half. From their beginnings in the 1950's and 1960's

† Some results in this paper were reported in conference papers: Roark, Sproat, and Shafran (2011) and Shafran et al. (2011). This research was supported in part by NSF Grants #IIS-0811745, #IIS-0905095 and #IIS-0964102, and DARPA grant #HR0011-09-1-0041. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the NSF or DARPA. We also thank Cyril Allauzen for useful discussion.
* Google Inc. 76 Ninth Ave, 4th Floor, New York, NY 10011, USA. E-mail: {rws,roark}@google.com
** Center for Spoken Language Understanding, Oregon Health & Science University, 3181 SW Sam Jackson Park Rd, GH40, Portland, OR 97239-3098, USA. Emails: {mahsa.yarmohamadi,zakshafran}@gmail.com

Submission received: 1st March, 2013. Revised version received: 5th November, 2013. Accepted for publication: 23rd December, 2013.

© 2012 Association for Computational Linguistics

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

to implement small hand-built grammars — e.g. Joshi and Hopely (1996) — through their applications in computational morphology in the 1980's (Koskenniemi 1983), finite-state models are now routinely applied in areas ranging from parsing (Abney 1996), to machine translation (Bangalore and Riccardi 2001; de Gispert et al. 2010), text normalization (Sproat 1996), and various areas of speech recognition including pronunciation modeling and language modeling (Mohri, Pereira, and Riley 2002).

The development of *weighted* finite state approaches (Mohri, Pereira, and Riley 2002; Mohri 2009) has made it possible to implement models that can rank alternative analyses. A number of weight classes — *semirings* — can be defined (Kuich and Salomaa 1986; Golan 1999), though for all practical purposes all actual applications use the *tropical semiring*, whose most obvious instantiation is as a way to combine *negative log probabilities* of words in a hypothesis in speech recognition systems. With few exceptions (e.g. (Eisner 2001)), there has been relatively little work on exploring applications of different semirings, in particular *structured semirings* consisting of tuples of weights.

In this paper we explore the use of what we term *lexicographic semirings*, tuples of weights where the comparison between a pair of tuples starts by comparing the first element of the tuple, then the second, and so forth until unequal values are found — just as lexicographic order is determined between words. We investigate two such lexicographic semirings, one based on pairs of tropical weights, and the other that uses a tropical weight paired with a novel string weight that we call the *categorial semiring*. The latter is based loosely on the operations of categorial grammar.

The first semiring we use to provide an exact encoding of language models as weighted finite-state transducers using epsilon arcs in place of failure arcs. The second we apply to the problem of selecting in a tagged lattice only the single-best tagging for each word sequence. In

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

each case we formally justify the application, and demonstrate the correctness and efficiency on real domains.

1.1 Definitions

Adopting the notations often used in the speech and language literature (Mohri 2009), a **semiring** is a 4-tuple $(K, \oplus, \otimes, \bar{0}, \bar{1})$ with a nonempty set K on which two binary operations are defined, namely the semiring plus $\oplus$ and semiring times $\otimes$, such that:

1. $(K, \oplus)$ is a commutative monoid with identity $\bar{0}$;
2. $(K, \otimes)$ is a monoid with identity $\bar{1}$;
3. $\otimes$ distributes over $\oplus$; and
4. $\bar{0} \otimes k = k \otimes \bar{0} = \bar{0} \quad \forall k \in K$.

Typically, $\bar{1} \neq \bar{0}$ is assumed, to avoid trivial semirings. The *tropical semiring* is an example of a well-known semiring and is defined as $(\Re \cup \{\infty\}, min, +, \infty, 0)$.

A **weighted finite-state transducer** T over a semiring $(K, \oplus, \otimes, \bar{0}, \bar{1})$ is an 8-tuple $(\Sigma, \Delta, Q, I, F, E, \lambda, \rho)$ where Σ and Δ are the finite input and output alphabets respectively, Q is a finite set of states of which I and F are initial and final subsets of states respectively, E is a finite set of transitions between pairs of states with an input and an output alphabet as well as a semiring weight $E \subseteq Q \times (\Sigma \cup \epsilon) \times (\Delta \cup \epsilon) \times K \times Q$, ϵ is an empty element in the alphabet, and λ and ρ are semiring weights associated with initial and final states respectively. A weighted finite-state acceptor can be regarded as a special case where either the input or the output alphabet is an empty set.

A weighted finite-state automaton (WFSA) or transducer is **deterministic** or **subsequential** if no two transitions leaving the same state have the same input label. A **generic determinization algorithm** can transform a weighted finite-state acceptor or transducer into its deterministic form if such a form exists. For details on the algorithm and conditions for determinization, see Section

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

6.2 in Mohri (2009). The condition most relevant for our purpose is that the algorithm works with any *weakly divisible* semiring. Briefly, a semiring $(K, \oplus, \otimes, \bar{0}, \bar{1})$ is said to be *divisible* if all non-$\bar{0}$ elements admit an inverse, that is, $(K - \bar{0})$ is a group. A semiring is *weakly divisible* if for any x and y in K such that $x \oplus y \neq \bar{0}$ there exists at least one z such that $(x \oplus y) \otimes z = x$. The $\otimes$ is *cancellative* if z is unique and can be written as $z = (x \oplus y)^{-1}x$. The non-unique case is not relevant here.

1.2 Lexicographic Semirings

The notion of *weight* can be extended to complex tuples of weights, and semirings over those tuples. Of interest to us here is a tuple-based semiring, the *lexicographic semiring*.

A $\langle W_1, W_2 \ldots W_n \rangle$-lexicographic weight is a tuple of weights where each of the weight classes $W_1, W_2 \ldots W_n$, must observe the *path property* (Mohri 2002). The path property of a semiring K is defined in terms of the *natural order* on K such that: $a <_K b$ iff $a \oplus b = a$. The tropical semiring mentioned above is a common example of a semiring that observes the path property, since

$$w_1 \oplus w_2 = \min\{w_1, w_2\}$$

and therefore if $w_1 <_K w_2$, then $w_1 \oplus w_2 = w_1$, and vice versa.

A particular instance of a lexicographic semiring, one that we will be making use of in this paper, involves a pair of tropical weights, which we will notate the $\langle \mathcal{T}, \mathcal{T} \rangle$-lexicographic semiring. For this semiring the operations $\oplus$ and $\otimes$ are defined as follows (Golan 1999, pp. 223–224):

$$\langle w_1, w_2 \rangle \oplus \langle w_3, w_4 \rangle = \begin{cases} \langle w_1, w_2 \rangle & \text{if } w_1 < w_3 \text{ or } (w_1 = w_3 \;\&\; w_2 < w_4) \\ \langle w_3, w_4 \rangle & \text{otherwise} \end{cases}$$

$$\langle w_1, w_2 \rangle \otimes \langle w_3, w_4 \rangle = \langle w_1 + w_3, w_2 + w_4 \rangle \tag{1}$$

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

The term "lexicographic" is an apt term for this semiring since the comparison for $\oplus$ is like the lexicographic comparison of strings, comparing the first elements, then the second, and so forth. Lexicographic semirings can be defined with other underlying semirings or tuple lengths.

1.3 An Example Application of Lexicographic Semiring: Implementing Ranking in Optimality Theory

As an example of a lexicographic semiring that has a tuple length (usually) greater than 2, consider one way in which one might implement constraint ranking in Optimality Theory.

Optimality Theory (Prince and Smolensky 2004) is a popular approach in phonology and other areas of linguistics. The basic tenet of the approach is that linguistic patterns are explained by a rank-ordered set of violable constraints. Actual forms generated via a function GEN, and selected by considering which of the forms violates the lowest ranked constraints. Each constraint may have multiple violations, but a single violation of a higher ranked constraint trumps any number of violations of a lower ranked constraint.

Consider the following recent example from `http://en.wikipedia.org/wiki/Optimality_theory#Example`, which accounts for the form of the regular noun plural suffix in English, which is voiceless /s/ after a voiceless stop (*cats*), /əz/ after a sibilant (*dishes*), and /z/ otherwise. Quoting directly from the Wikipedia example, the following constraints in the order given account for the phenomena:

1. ***SS** - Sibilant-Sibilant clusters are ungrammatical: one violation for every pair of adjacent sibilants in the output.
2. **Agree(Voi)** - Agree in specification of [voi]: one violation for every pair of adjacent obstruents in the output which disagree in voicing.
3. **Max** - Maximize all input segments in the output: one violation for each segment in the input that doesn't appear in the output. (This constraint prevents deletion.)

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

4. **Dep** - Output segments are dependent on having an input correspondent: one violation for each segment in the output that doesn't appear in the input. (This constraint prevents insertion.)
5. **Ident(Voi)** - Maintain the identity of the [voi] specification: one violation for each segment that differs in voicing between the input and output.

Consider the example of *dishes*. From a presumed underlying form of **dish+z**, GEN generates a range of possible forms, including those in the lefthand column in the table below:

dish+z	*SS	Agree	Max	Dep	Ident
☞ dishiz				*	
dishis				*	*!
dishz	*!	*			
dish			*!		
dishs	*!				*

Asterisks indicate violations, and exclamation marks the critical violation that rules out the particular form. Both *dishs* and *dishz* have violations of ***SS**, and since none of the other forms violate ***SS**, and ***SS** is highest ranked, those two violations are critical. Concomitantly any other violations — e.g. *dishs* violation of **Ident** — are irrelevant for determining the fate of those forms. Moving down the constraint hierarchy, *dish* violates **Max**, since the suffix does not appear in this form; again this violation is critical, since the remaining two forms do not violate the constraint. Both *dishis* and and *dishiz* violate **Dep** since there's an inserted segment and are thus equally bad according to that constraint. So to decide between the two forms, we go to the next lower constraint, **Ident(Voi)**, which *dishis* violates because the underlying *z* is changed to an *s*. This violation is therefore critical, and the winning form is *dishiz*, indicated by the right-pointing hand.

There have been many finite-state models of Optimality Theory (Ellison 1994; Eisner 1998; Albro 1998; Frank and Satta 1998; Karttunen 1998; Eisner 2000), and our point here is not to provide a fully worked out implementation of the model. Rather we wish to show that an appropriately defined lexicographic semiring can readily model the constraint ranking.

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

We start by defining the Violation Semiring $\mathcal{V}$ as $(\mathbb{Z} \cup \{\infty\}, min, +, \infty, 0)$; $\mathcal{V}$ is clearly just a special case of the Tropical Semiring where the values of the weights are restricted to be non-negative integers. We then define the Optimality Semiring O as $\langle \mathcal{V}, \mathcal{V}, \dots \rangle$, namely a lexicographic tuple over $\mathcal{V}$. The number of elements of the tuple is the same as the number of constraints needed in the system being described. If there are five rank-ordered constraints, as above, then $\langle \mathcal{V}, \mathcal{V}, \dots \rangle$ is a 5-tuple over $\mathcal{V}$.

Assuming that the GEN function generates a lattice S of possible surface forms for a word, and a set of n constraints, we need a set of constraint acceptors $C_1 \dots C_n$, each of which matches individual violations of the constraints, and where each violation of C_i is weighted as $\langle 0, 0, \dots, 0, 1, 0, \dots, 0 \rangle$, with 1 in the ith position in the weight. So in the given example above, ***SS** would be a finite-state acceptor that allows sibilant-sibilant sequences, but only at a cost $\langle 1, 0, 0, 0, 0 \rangle$ per sequence. Assuming that when GEN deletes an element (as in the form *dish*), it marks the deletion (e.g. *dish**) then we can implement **Max** as an acceptor that accepts the deletion symbol with cost $\langle 0, 0, 1, 0, 0 \rangle$ per instance. In a similar vein, assuming that any *inserted* elements are marked (e.g. *dish>iz*), then **Dep** will allow the insertion marker with cost $\langle 0, 0, 0, 1, 0 \rangle$ per instance. Finally, **Ident(Voi)** assumes that a change in voicing is marked somehow (e.g. *dishis<*), and this marker will be accepted with cost $\langle 0, 0, 0, 0, 1 \rangle$ per instance.

Given the lattice of forms S, the optimal form will be obtained by intersecting S with each of the constraints, and then computing the shortest path to select the form with the best overall cost. Formally:

$$\text{ShortestPath}[S \cap \bigcap_{i=0}^{n} C_i] \tag{2}$$

In the case at hand, the costs of each of the paths will be as follows, ranked from worst to best, from which it immediately can be seen that the optimal form is *dishiz*:

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

dishz	$\langle 1, 1, 0, 0, 0 \rangle$
dishs	$\langle 1, 0, 0, 0, 1 \rangle$
dish	$\langle 0, 0, 1, 0, 0 \rangle$
dishis	$\langle 0, 0, 0, 1, 1 \rangle$
dishiz	$\langle 0, 0, 0, 1, 0 \rangle$

Hence a lexicographic semiring designed for Optimality Theory would have as many dimensions as constraints in the grammar[1]. In what follows, we discuss two specific binary lexicographic semirings of utility for encoding and performing inference with sequence models encoded as weighted finite-state transducers.

2. Paired tropical lexicographic semiring and applications

We start in this section with a simple application of a paired tropical-tropical lexicographic semiring to the problem of representing failure ('ϕ') transitions in an n-gram language model. While ϕ-transitions can be represented exactly, as we shall argue below, there are limitations on their use, limitations that can be overcome by representing them instead as ϵ arcs and lexicographic weights.

2.1 Lexicographic Language Model Semiring

Representing smoothed n-gram language models as weighted finite-state transducers (WFST) is most naturally done with a *failure transition*, which reflects the semantics of the "otherwise" formulation of smoothing (Allauzen, Mohri, and Roark 2003). For example, the typical backoff formulation of the probability of a word w given a history h is as follows

$$\mathrm{P}(w \mid h) = \begin{cases} \overline{\mathrm{P}}(w \mid h) & \text{if } c(hw) > 0 \\ \alpha_h \mathrm{P}(w \mid h') & \text{otherwise} \end{cases} \tag{3}$$

where $\overline{\mathrm{P}}$ is an empirical estimate of the probability that reserves small finite probability for unseen n-grams; α_h is a backoff weight that ensures normalization; and h' is a backoff history

1 For partial orderings, where multiple constraints are at the same level in the absolute dominance hierarchy, just one dimension would be required for all constraints at the same level.

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

typically achieved by excising the earliest word in the history h. The principal benefit of encoding the WFST in this way is that it only requires storing n-gram transitions explicitly for observed n-grams, i.e., count greater than zero, as opposed to all possible n-grams of the given order which would be infeasible in, for example, large vocabulary speech recognition. This is a massive space saving, and such an approach is also used for non-probabilistic stochastic language models, such as those trained with the perceptron algorithm (Roark, Saraclar, and Collins 2007), as the means to access all and exactly those features that should fire for a particular sequence in a deterministic automaton. Similar issues hold for other finite-state sequence processing problems, e.g., tagging, bracketing or segmenting, as with the POS tagger that we use for experimental results in Section 3.4.

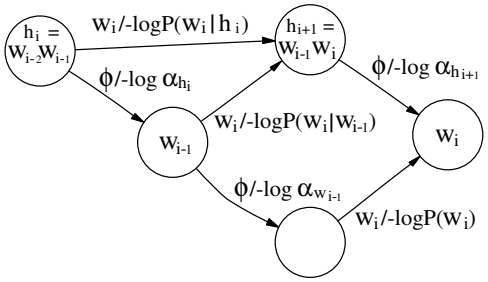
Failure transitions are an implicit method for representing a much larger explicit automaton – in the case of n-gram models, all possible n-grams for that order. During composition with the model, the failure transition must be interpreted on the fly, keeping track of those symbols that have already been found leaving the original state, and only allowing failure transition traversal for symbols that have not been found (the semantics of "otherwise"). This compact implicit representation cannot generally be preserved when composing with other models, e.g., when combining a language model with a pronunciation lexicon as in widely-used FST approaches to speech recognition (Mohri, Pereira, and Riley 2002). Moving from implicit to explicit representation when performing such a composition leads to an explosion in the size of the resulting transducer, frequently making the approach intractable. In practice, an off-line approximation to the model is made, typically by treating the failure transitions as epsilon transitions (Mohri, Pereira, and Riley 2002; Allauzen, Mohri, and Roark 2003), allowing large transducers to be composed and optimized off-line. These complex approximate transducers are then used during first-pass decoding, and the resulting pruned search graphs (e.g., word lattices) can be rescored with exact language models encoded with failure transitions. Failure transitions can be used

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

to exactly encode a wide range of language models, including class-based language models (Allauzen, Mohri, and Roark 2003) or discriminatively trained n-gram language models (Roark, Saraclar, and Collins 2007) – allowing for full lattice rescoring rather than n-best list extraction.

Similar problems arise when building, say, POS taggers as WFSTs: not every POS tag sequence will have been observed during training, hence failure transitions will achieve great savings in the size of models. Yet discriminative models may include complex features that combine both input stream (word) and output stream (tag) sequences in a single feature, yielding complicated transducer topologies for which effective use of failure transitions may not be possible. An exact encoding using other mechanisms is required in such cases to allow for off-line representation and optimization.

Figure 1
Deterministic finite-state representation of n-gram models with negative log probabilities (tropical semiring). The symbol ϕ labels backoff transitions. Modified from Roark and Sproat (2007), Figure 6.1.



2.1.1 Standard encoding. For language model encoding, we will differentiate between two classes of transitions: *backoff arcs* (labeled with a ϕ for failure, or with ϵ using our new semiring); and *n-gram arcs* (everything else, labeled with the word whose probability is assigned). Each state in the automaton represents an n-gram history string h and each n-gram arc is weighted with the (negative log) conditional probability of the word w labeling the arc given the history h. We assume that, for every n-gram hw explicitly represented in the language model, every proper prefix and every proper suffix of that n-gram is also represented in the model. Hence, if h is a

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

state in the model, then h' (the suffix of h of length $|h|-1$) will also be a state in the model. For a given history h and n-gram arc labeled with a word w, the destination of the arc is the state associated with the longest suffix of the string hw that is a history in the model. This will depend on the Markov order of the n-gram model. For example, consider the trigram model schematic shown in Figure 1, in which only history sequences of length 2 are kept in the model. Thus, from history $h_i = w_{i-2}w_{i-1}$, the word w_i transitions to $h_{i+1} = w_{i-1}w_i$, which is the longest suffix of $h_i w_i$ in the model.

As detailed in the "otherwise" semantics of equation 3, backoff arcs transition from state h to a state h', typically the suffix of h of length $|h| - 1$, with weight $(-\log \alpha_h)$. We call the destination state a backoff state. This recursive backoff topology terminates at the unigram state, i.e., $h = \epsilon$, no history.

Backoff states of order k may be traversed either via ϕ-arcs from the higher order n-gram of order $k + 1$ or via an n-gram arc from a lower order n-gram of order $k - 1$. This means that no n-gram arc can enter the zeroeth order state (final backoff), and full-order states — history strings of length $n - 1$ for a model of order n — may have n-gram arcs entering from other full-order states as well as from backoff states of history size $n - 2$.

2.1.2 Exact encoding of a backoff model with lexicographic language model semiring. For an LM machine M on the tropical semiring with failure transitions, we can simulate ϕ-arcs in a standard LM topology by a topologically equivalent machine M' on the lexicographic $\langle \mathcal{T}, \mathcal{T} \rangle$ semiring, where ϕ has been replaced with epsilon, as follows. Let s_i and s'_i be equivalent states in M and M' respectively. For every n-gram arc with label w and weight c, source state s_i and destination state s_j, construct an n-gram arc with label w, weight $\langle 0, c \rangle$, source state s'_i, and destination state s'_j. The exit cost of each state is constructed as follows. If the state is non-final, $\langle \infty, \infty \rangle$. Otherwise if it final with exit cost c it will be $\langle 0, c \rangle$.

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

The pseudocode for converting a failure encoded language model into lexicographic language model semiring is enumerated in Figure 2 and illustrated in Figure 3.

ADDARC(L, s_1, Arc(s_2, l_i, l_o, w))

```
1  add arc to Arcs(s_1, L)
2  next-state(arc) ← s_2
3  in-label(arc) ← l_i
4  out-label(arc) ← l_o
5  weight(arc) ← w
```

CONVERT2LEXLM(L)

```
1  n ← max_{s in States(L)} length(history(s))
2  L' ← new FST
3  for s in States(L) do
4      add state s' to L'
5      if s is Start(L) then                          ▷ If (unique) start state
6          Start(L') ← s'
7      if Final(s, L) = ∞ then                        ▷ If state not final
8          Final(s', L') ← ⟨∞, ∞⟩
9      else Final(s', L') ← ⟨0, Final(s, L)⟩
10     for arc in Arcs(s, L) do
11         if in-label(arc) = φ then                  ▷ If backoff arc
12             k ← length(history(next-state(arc)))
13             ADDARC(L', s', Arc(next-state(arc)', ϵ, ϵ, ⟨Φ^⊗(n−k), weight(arc)⟩))
14         else ADDARC(L', s', Arc(next-state(arc)', in-label(arc), out-label(arc), ⟨0, weight(arc)⟩))
15 return L'
```

Figure 2
Pseudocode for converting an n-gram failure language model into an equivalent lexicographic language model acceptor. The states have an associated history whose length depends on the degree of backoff.

Let n be the length of the longest history string in the model. For every ϕ-arc with (backoff) weight c, source state s_i, and destination state s_j representing a history of length k, construct an ϵ-arc with source state s'_i, destination state s'_j, and weight $\langle \Phi^{\otimes(n-k)}, c \rangle$, where $\Phi > 0$ and $\Phi^{\otimes(n-k)}$ takes Φ to the $(n-k)^{\text{th}}$ power with the $\otimes$ operation. In the tropical semiring, $\otimes$ is $+$, so $\Phi^{\otimes(n-k)} = (n-k)\Phi$. For example, in a trigram model, if we are backing off from a bigram state h (history length = 1) to a unigram state, $n - k = 2 - 0 = 2$, so we set the backoff weight to $\langle 2\Phi, -\log \alpha_h \rangle$ for some $\Phi > 0$. In the special case where the ϕ-arc has weight ∞, which can happen in some language model topologies, the corresponding $\langle \mathcal{T}, \mathcal{T} \rangle$ weight will be $\langle \infty, \infty \rangle$.

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

In order to combine the model with another automaton or transducer, we would need to also convert those models to the $\langle \mathcal{T}, \mathcal{T} \rangle$ semiring. For these automata, we simply use a default transformation such that every transition with weight c is assigned weight $\langle 0, c \rangle$. For example, given a word lattice L, we convert the lattice to L' in the lexicographic semiring using this default transformation, and then perform the intersection $L' \cap M'$. By removing epsilon transitions and determinizing the result, the low cost path for any given string will be retained in the result, which will correspond to the path achieved with ϕ-arcs. Finally we project the second dimension of the $\langle \mathcal{T}, \mathcal{T} \rangle$ weights to produce a lattice in the tropical semiring, which is equivalent to the result of $L \cap M$, i.e.,

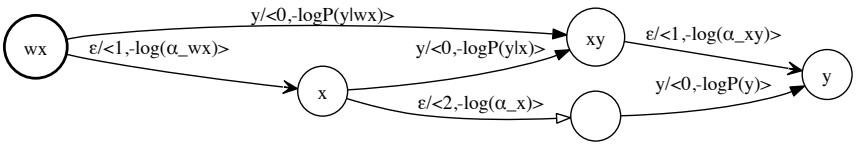
$$\mathcal{C}_2(\textbf{det}(\textbf{eps-rem}(L' \cap M'))) = L \cap M \tag{4}$$

where $\mathcal{C}_2$ denotes projecting the second-dimension of the $\langle \mathcal{T}, \mathcal{T} \rangle$ weights, **det**$(\cdot)$ denotes determinization, and **eps-rem**$(\cdot)$ denotes ϵ-removal.

2.2 Proof of Equivalence

We wish to prove that for any machine N, $\mathrm{ShortestPath}(M' \cap N')$ passes through the equivalent states in M' to those passed through in M for $\mathrm{ShortestPath}(M \cap N)$. Therefore determinization of the resulting intersection after ϵ-removal yields the same topology as intersection

Figure 3
An example to illustrate the encoding of lexicographic language model semiring, where we set Φ to 1. This is an instance of the general trigram LM depicted in Figure 1 with the sequence $w_{i-2}w_{i-1}w_i = wxy$. The scalar negative log probabilities are transformed from tropical semiring into tuples as explained in the text. The solid full, open, and unfilled full arrowheads correspond to the three cases – no backoff, bigram backoff and unigram backoff, respectively.



Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

with the equivalent ϕ machine. Intuitively, since the first dimension of the $\langle \mathcal{T}, \mathcal{T} \rangle$ weights is 0 for n-gram arcs and > 0 for backoff arcs, the shortest path will traverse the fewest possible backoff arcs; further, since higher-order backoff arcs cost less in the first dimension of the $\langle \mathcal{T}, \mathcal{T} \rangle$ weights in M', the shortest path will include n-gram arcs at their earliest possible point.

We prove this by induction on the state-sequence of the path p/p' up to a given state s_i/s'_i in the respective machines M/M'.

Base case: If p/p' is of length 0, and therefore the states s_i/s'_i are the initial states of the respective machines, the proposition clearly holds.

Inductive step: Now suppose that p/p' visits $s_0...s_i$/$s'_0...s'_i$ and we have therefore reached s_i/s'_i in the respective machines. Suppose the cumulated weights of p/p' are W and $\langle \Psi, W \rangle$, respectively. We wish to show that whichever s_j is next visited on p (i.e., the path becomes $s_0...s_i s_j$) the equivalent state s' is visited on p' (i.e., the path becomes $s'_0...s'_i s'_j$).

Let w be the next symbol to be matched leaving states s_i and s'_i. There are four cases to consider:

1. there is an n-gram arc leaving states s_i and s'_i labeled with w, but no backoff arc leaving the state;
2. there is no n-gram arc labeled with w leaving the states, but there is a backoff arc;
3. there is no n-gram arc labeled with w and no backoff arc leaving the states; and
4. there is both an n-gram arc labeled with w and a backoff arc leaving the states.

In cases (1) and (2), there is only one possible transition to take in either M or M', and based on the algorithm for construction of M' given in Section 2.1.2, these transitions will point to s_j and s'_j respectively. Case (3) leads to failure of intersection with either machine. This leaves case (4) to consider. In M, since there is a transition leaving state s_i labeled with w, the backoff arc, which is a failure transition, cannot be traversed, hence the destination of the n-gram arc s_j will

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

be the next state in p. However, in M', both the n-gram transition labeled with w and the backoff transition, now labeled with ϵ, can be traversed. What we will now prove is that the shortest path through M' cannot include the backoff arc in this case.

In order to emit w by taking the backoff arc out of state s'_i, one or more backoff (ϵ) transitions must be taken, followed by an n-gram arc labeled with w. Let k be the order of the history represented by state s'_i, hence the cost of the first backoff arc is $\langle (n-k)\Phi, -\log(\alpha_{s'_i})\rangle$ in our semiring. If we traverse m backoff arcs prior to emitting the w, the first dimension of our accumulated cost will be $m(n-k+\frac{m-1}{2})\Phi$, based on our algorithm for the construction of M' given in Section 2.1.2. Let s'_l be the destination state after traversing m backoff arcs followed by an n-gram arc labeled with w. Note that, by definition, $m \leq k$, and $k-m+1$ is the order of state s'_l. Based on the construction algorithm, the state s'_l is also reachable by first emitting w from state s'_i to reach state s'_j followed by some number of backoff transitions, as can be seen from the paths between state w_{i-1} and w_i in the trigram model schematic in Figure 1. The order of state s'_j is either k (if k is the highest order in the model) or $k+1$ (by extending the history of state s'_i by one word). If it is of order k, then it will require $m-1$ backoff arcs to reach state s'_l, one fewer than the path to state s'_l that begins with a backoff arc, for a total cost of $(m-1)(n-k+\frac{m-1}{2})\Phi$ which is less than $m(n-k+\frac{m-1}{2})\Phi$. If state s'_j is of order $k+1$, there will be m backoff arcs to reach state s'_l, but with a total cost of $m(n-(k+1)+\frac{m-1}{2})\Phi = m(n-k+\frac{m-3}{2})\Phi$ which is also less than $m(n-k+\frac{m-1}{2})\Phi$. Hence the state s'_l can always be reached from s'_i with a lower cost through state s'_j than by first taking the backoff arc from s'_i. Therefore the shortest path on M' must follow $s'_0 \ldots s'_i s'_j$. □

This completes the proof.

2.3 Experimental Comparison of ϵ, ϕ and $\langle \mathcal{T}, \mathcal{T} \rangle$ encoded language models

For our experiments we used lattices derived from a very large vocabulary continuous speech recognition system, which was built for the 2007 GALE Arabic speech recognition task, and

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

used in the work reported in Lehr and Shafran (2011). The lexicographic semiring was evaluated on the development set (2.6 hours of broadcast news and conversations; 18K words). The 888 word lattices for the development set were generated using a competitive baseline system with acoustic models trained on about 1000 hrs of Arabic broadcast data and a 4-gram language model. The language model consisting of 122M n-grams was estimated by interpolating 14 components. The vocabulary is relatively large at 737K and the associated dictionary has only single pronunciations.

The language model was converted to the automaton topology described earlier, using OpenFst (Allauzen et al. 2007), and represented in three ways: first as an approximation of a failure machine using epsilons instead of failure arcs; second as a correct failure machine; and third using the lexicographic construction derived in this paper. Note that all of these options are available for representing language models in the OpenGrm library (Roark et al. 2012).

The three versions of the LM were evaluated by intersecting them with the 888 lattices of the development set. The overall error rate for the systems was 24.8%—comparable to the state-of-the-art on this task[2]. For the shortest paths, the failure and lexicographic machines always produced identical lattices (as determined by FST equivalence); in contrast, 78.6% of the shortest paths from the epsilon approximation are different, at least in terms of weights, from the shortest paths using the failure LM. For full lattices 6.1% of the lexicographic outputs differ from the failure LM outputs, due to small floating point rounding issues; 98.9% of the epsilon approximation outputs differ[3].

2 The error rate is a couple of points higher than in Lehr and Shafran (2011) since we discarded non-lexical words, which are absent in maximum likelihood estimated language model and are typically augmented to the unigram backoff state with an arbitrary cost, fine-tuned to optimize performance for a given task.

3 The very slight differences in these percentages (less than 3% absolute in all cases) versus those originally reported in Roark, Sproat, and Shafran (2011) are due to small changes in conversion from ARPA format language models to OpenFst encoding in the OpenGrm library (Roark et al. 2012), related to ensuring that, for every n-gram explicitly included in the model, every proper prefix and proper suffix is also included in the model, something that the ARPA format does not require.

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

In terms of size, the failure LM, with 5.7 million arcs requires 97 Mb. The equivalent $\langle \mathcal{T}, \mathcal{T} \rangle$-lexicographic LM requires 120 Mb, due to the doubling of the size of the weights.[4] To measure speed, we performed the intersections 1000 times for each of our 888 lattices on a 2993 MHz Intel® Xeon® CPU, and took the mean times for each of our methods. The 888 lattices were processed with a mean of 1.62 seconds in total (1.8 msec per lattice) using the failure LM; using the $\langle \mathcal{T}, \mathcal{T} \rangle$-lexicographic LM required 1.8 seconds (2.0 msec per lattice), and is thus about 11% slower. Epsilon approximation, where the failure arcs are approximated with epsilon arcs took 1.17 seconds (1.3 msec per lattice). The slightly slower speeds for the exact method using the failure LM, and $\langle \mathcal{T}, \mathcal{T} \rangle$ are due to the overhead of (1) computing the failure function at runtime for the failure LM, and (2) determinization for the $\langle \mathcal{T}, \mathcal{T} \rangle$ representation. After intersection (and determinization, if required), there is no size difference in the lattices resulting from any of the three methods.

In this section we have shown that the failure-arc representation of backoff in a finite-state language model topology can be exactly represented using ϵ arcs, and weights in the $\langle \mathcal{T}, \mathcal{T} \rangle$ lexicographic semiring.

We turn in the next section to another application of lexicographic semirings, this time involving a novel string semiring as one of the components.

3. Tagging determinization on lattices

In many applications of speech and language processing, we generate intermediate results in the form of a lattice to which we apply finite-state operations. For example, we might POS tag the words in an ASR output lattice as an intermediate stage for detecting out-of-vocabulary nouns. This involves composing the lattices with a POS tagger and will result in a weighted transducer that maps from input words to tags.

4 If size became an issue, the first dimension of the $\langle \mathcal{T}, \mathcal{T} \rangle$-weight can be represented by a single byte.

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

Suppose we want from that transducer all the recognized word sequences, but with each word sequence just the single-best tagging. One obvious way to do this would be to extract sublattices containing all possible taggings of each word sequence, compute the shortest path of each such sublattice, and union the results back together. There are various ways this might be accomplished algorithmically, but in general it will be an expensive operation.

With a little thought it will be clear that at an appropriate level of abstraction the problem we have just described involves determinization. That is, the result is deterministic in the sense that for any input, there is a unique path through the lattice. But one cannot simply apply transducer determinization since, for one reason, any given input may have multiple outputs and thus is non-functional and not even p-subsequential (Mohri 2009).

In this section we describe two methods, both of which make use of novel weight classes consisting of a pair of a tropical weight and a string weight, which allow a solution that involves determinization on an *acceptor* in that semiring. One, due to Povey and colleagues (Povey et al. 2012) is described in Section 3.1. Our own work, also previously reported in Shafran et al. (2011), is presented in Sections 3.2 and 3.3. In Section 3.4 we compare the approaches for efficiency.

3.1 Povey's approach

Povey et al. (2012) define an appropriate pair weight structure such that determinization yields the single-best path for all unique sequences. In their pair weight (T, S), T is the original (tropical) weight in the lattice, and S is a form of string weight representing the tags. Using here the more formal '$\cdot$' to denote concatenation, they define $\oplus$ and $\otimes$ operations as:

$$(w_1, w_2) \oplus (w_3, w_4) = \begin{cases} (w_1, w_2) \text{ if } w_1 < w_3; \text{ else} \\ (w_3, w_4) \text{ if } w_1 > w_3; \text{ else} \\ (w_1, w_2) \text{ if } |w_2| < |w_4|; \text{ else} \\ (w_3, w_4) \text{ if } |w_2| > |w_4|; \text{ else} \\ (w_1, w_2) \text{ if } w_2 <_L w_4; \text{ else} \\ (w_3, w_4) \end{cases}$$

$$(w_1, w_2) \otimes (w_3, w_4) = (w_1 + w_3, w_2 \cdot w_4) \quad (5)$$

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

Here $|w_i|$ denotes the length of the sequence w_i. The $\oplus$ of two pair weights in this definition does not necessarily left-divide the weights, so the standard definition of determinization does not work on this semiring. They change the standard determinization of a lexicographic semiring by defining a new "common divisor" operation $\boxplus$ for their pair weight. In the standard determinization, $\oplus$ operation finds the common divisor of the weights.

$$(w_1, w_2) \boxplus (w_3, w_4) = (w_1 \oplus w_3, \mathit{LongestCommonPrefix}(w_2, w_4)) \tag{6}$$

Povey et al. describe their method in the context of an exact lattice generation task. They create a state-level lattice during ASR decoding and determinize it to retain only the best-scoring path for each word sequence. They invert the state-level lattice, encode it as an acceptor with its input label equal to the input label of lattice (word), and the pair weight equal to the weight and output label of the lattice, and finally determinize the acceptor to get the best state-level alignment for each word sequence.

For efficiency reasons, determinization and epsilon removal – which is optimized for this particular type of weight – are done simultaneously in their method. For the string part, they use a data structure involving a hash table which enables string concatenation in linear time.

3.2 Categorial Semiring

An alternative approach to that of Povey and colleagues eschews a special definition of determinization, using instead the standard definition already provided in the OpenFst library. To this end, we designed a lexicographic weight pair that incorporates a tropical weight as the first dimension and a novel form of string weight for the second dimension to represent the tags. Note that the standard string weight (for example that implemented in the OpenFst library) will not do. In that semiring, $w_1 \otimes w_2$ is defined as concatenation; and $w_1 \oplus w_2$ is defined as the longest common prefix of w_1 and w_2, which is not in general equal to either w_1 or w_2. Thus the string

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

weight class does not have the path property, and hence it cannot be used as an element of a lexicographic semiring tuple.

We can solve that problem by having $w_1 \oplus w_2$ be the lexicographic minimum (according to some definition of string ordering) of w_1 and w_2, which will guarantee that the semiring has the path property. But now we need a way to make the semiring *weakly divisible*, so that when weights are pushed during the determinization operation, the "loser" can be preserved. For a string weight, this can be achieved by recording the division so that a subsequent $\otimes$ operation with the appropriate (inverse) string is *cancellative*. Thus if $x \oplus y = y$, then there should be a $z = y \backslash x$, such that $(x \oplus y) \otimes z = (x \oplus y) \otimes y \backslash x = y \otimes y \backslash x = x$

A natural model for this is *categorial grammar* (Lambek 1958). In categorial grammar, there are a set of primitive categories, such as *N*, *V*, *NP*, as well as a set of complex types constructed out of left ($\backslash$) or right ($/$) division operators. An expression $X \backslash Y$ denotes a category that, when combined with an X on its left, makes a Y. For example, a verb phrase in English could be represented as a *NP*$\backslash$*S*, since when combined with an *NP* on the left, it makes an *S*. Similarly a determiner is *NP*$/$*N*, since it combines with an *N* on the right to make an *NP*.

A *categorial semiring* can be defined for both left- and righthand versions. We restrict ourselves in this discussion to the *left-categorial* semiring, the *right-categorial* version being equivalently defined. Thus we define the *left-categorial* semiring $(\Sigma^*, \oplus, \otimes, \infty_s, \epsilon)$ over strings Σ^* with ϵ and ∞_s as special infinity and null string symbols respectively (as in the normal string semiring). The $\otimes$ operation accumulates the symbols along a path using standard string concatenation. The $\oplus$ operation simply involves a string comparison between the string representations of (possibly accumulated versions of) the output symbols or tags using lexicographic less-than ($<_L$). The $\oslash$ operation records the left-division in the same sense as categorial grammar. Finally we introduce a function REDUCE, which performs reductions on any string, so that for example

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

$\textsc{Reduce}(a \cdot a\backslash b) = b$:

$$w_1 \oplus w_2 = \begin{cases} w_1 \text{ if } w_1 <_L w_2 \\ w_2 \text{ otherwise} \end{cases}$$

$$w_1 \otimes w_2 = \textsc{Reduce}(w_1 \cdot w_2)$$

$$w_1 \oslash w_2 = w_2\backslash w_1 \tag{7}$$

We further define grouping brackets $\langle$ and $\rangle$ as part of the notation so that for example a complex weight $a\backslash bc$ divided into d is $\langle a\backslash bc\rangle\backslash d$.

Unfortunately, while the above definition is close to what we want, it is not a semiring, because with that definition, $\otimes$ is not distributive over $\oplus$. As stated in Section 1.1, a semiring must be defined in such a way that $w_1 \otimes (w_2 \oplus w_3) = (w_1 \otimes w_2) \oplus (w_1 \otimes w_3)$. To see that this is not in general the case with the above definition, let $w_1 = c$, $w_2 = c\backslash a$ and $w_3 = b$. Using '_' to indicate concatenation of two weights, and assuming that $a <_L b <_L c$, then:

$$c \otimes (c\backslash a \oplus b) = c \otimes b = c\_b \tag{8}$$

whereas

$$(c \otimes c\backslash a) \oplus (c \otimes b) = a \oplus c\_b = a \tag{9}$$

To solve this problem requires modifying our semiring definition slightly to distinguish between the *history*, denoted as **h**, and the *value*, denoted as **v**. The history records the concatenations involved in creating the particular weight instance, without any concomitant reductions, whereas the value is the actual value of the weight, including the reductions. We redefine the left categorial

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

semiring as follows:

$$w_1 \oplus w_2 = \begin{cases} w_1 \text{ if } \mathbf{h}(w_1) <_L \mathbf{h}(w_2) \\ w_2 \text{ otherwise} \end{cases}$$

$$w_1 \otimes w_2 = w_3, \text{where } \mathbf{h}(w_3) = \mathbf{h}(w_1) \cdot \mathbf{h}(w_2) \text{ and } \mathbf{v}(w_3) = \textsc{Reduce}(\mathbf{h}(w_3))$$

$$w_1 \oslash w_2 = w_3, \text{where } \mathbf{h}(w_3) = \mathbf{h}(w_2) \backslash \mathbf{h}(w_1) \text{ and } \mathbf{v}(w_3) = \mathbf{v}(w_2) \backslash \mathbf{v}(w_1) \tag{10}$$

Note that the history now defines the natural ordering of the semiring. Returning to the problematic case above we note that it is still the case that $c \otimes (c\backslash a \oplus b) = c \otimes b = c\_b$. This is because for $c\backslash a \oplus b$, $\mathbf{h}(b) <_L \mathbf{h}(c\backslash a)$, so that $c\backslash a \oplus b = b$. For, $(c \otimes c\backslash a) \oplus (c \otimes b)$, however, we now get the same result. $(c \otimes b)$ has both a history and a value of c_b. $(c \otimes c\backslash a)$, on the other hand, has a value of a as before, but a history of $c\_c\backslash a$. The sum of these weights is determined by the lexicographic comparison $c\_b <_L c\_c\backslash a$, and thus $(c \otimes c\backslash a) \oplus (c \otimes b) = c\_b$

The value of $\otimes$ is defined as the reduction of the *history* of the concatenated weight histories rather than the concatenated weight values in order to guarantee that $\otimes$ is associative: for semiring $\otimes$ it must be the case that $w_1 \otimes (w_2 \otimes w_3) = (w_1 \otimes w_2) \otimes w_3$. Let $w_1 = a$, $w_2 = a\backslash b$ and $w_3 = \langle a\backslash b\rangle\backslash c$. If we compute the values of the multiplications on the basis of the values of the weights we have

$$a \otimes (a\backslash b \otimes \langle a\backslash b\rangle\backslash c) = a \otimes c = a\_c \tag{11}$$

but

$$(a \otimes a\backslash b) \otimes \langle a\backslash b\rangle\backslash c = b \otimes \langle a\backslash b\rangle\backslash c = b\langle a\backslash b\rangle\backslash c \tag{12}$$

However, the histories in both cases are given as:

$$\begin{aligned} a \otimes (a\backslash b \otimes \langle a\backslash b\rangle\backslash c) &= a\_a\backslash b\_\langle a\backslash b\rangle\backslash c \\ &= (a \otimes a\backslash b) \otimes \langle a\backslash b\rangle\backslash c \end{aligned} \tag{13}$$

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

The value $\mathbf{v}(a\_a\backslash b\_\langle a\backslash b\rangle\backslash c)$, if we follow a greedy right-to-left reduction, becomes a_c.

Note that one difference between the categorial semiring and standard categorial grammar is that in the categorial semiring division may involve complex categorial weights that are themselves concatenated, as we have already seen. For example one may need to left-divide a category *NN* by a complex category that itself involves a division and a multiplication. We might thus produce a category such as $\langle$*VB*$\backslash$*JJ_NN*$\rangle\backslash$*NN*. We assume division has precedence over times (concatenation), so in order to represent this complex category, the disambiguating brackets $\langle\rangle$ are needed. The interpretation of this category is something that, when combined with the category *VB*$\backslash$*JJ_NN* on the left, makes an *NN*.

3.3 Implementation of tagging determinization using a lexicographic semiring

Having chosen the semirings for the first and second weights in the transformed WFSA, we now need to define a joint semiring over both the weights and specify its operation. For this we return to the lexicographic semiring. Specifically we define the $\langle\mathcal{T},\mathcal{C}\rangle$ *Lexicographic Semiring* $(\langle\Re\cup\{\infty\},\Sigma^*\rangle,\oplus,\otimes,\bar{0},\bar{1})$ over a tuple of tropical and left-categorial weights, inheriting their corresponding identity elements. The $\bar{0}$ and $\bar{1}$ elements for the categorial component are defined the same way as in the standard string semiring namely, respectively, as the infinite string, and as the empty string ϵ, discussed above.

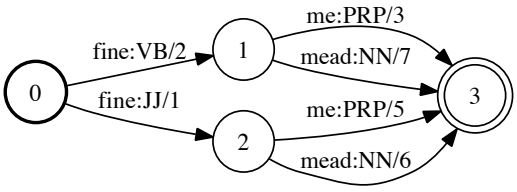


Figure 4
A simple example for illustrating the application of the $\langle\mathcal{T},\mathcal{C}\rangle$-lexicographic semiring, plus determinization, for finding the single best tagging for each word sequence. Note that a simple application of the shortest path to this example would discard all analyses of *fine mead*.

A Sketch of a Proof of Correctness: The correctness of this lexicographic semiring, combined with determinization, for our problem could be shown by tracing the results of operation in a

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

generic determinization algorithm, as in Mohri (2009). Instead, here we provide an intuition using the example in Figure 4. The two input strings *fine me* and *fine mead* share the prefix *fine*. In the first case *fine* is a verb (VB), whereas in the second it is a adjective (JJ). When two outgoing arcs have same input symbols, the determinization algorithm choses the arc with the lowest weight, $\langle 1, JJ \rangle$. For potential future use the other weight $\langle 2, VB \rangle$ is divided by the lowest weight $\langle 1, JJ \rangle$ and the result $\langle 1, JJ \backslash VB \rangle$ is saved. (Note that the divide operation for the tropical semiring is arithmetic subtraction.) When processing the next set of arcs, the determinization algorithm will encounter two paths for the input *fine mead*. The accumulated weight on the path through nodes 0-2-3 is straightforward and is $\langle 1, JJ \rangle \otimes \langle 6, NN \rangle = \langle 7, JJ\_NN \rangle$. The accumulated weight computed by the determinization algorithm through 0-1-3 consists of three components – the lowest weight for *fine*, the saved residual and the arc weight for *mead* from 1-3. Thus, the accumulated weight for 0-1-3 for *fine mead* is $\langle 1, JJ \rangle \otimes \langle 1, JJ \backslash VB \rangle \otimes \langle 7, NN \rangle = \langle 9, VB\_NN \rangle$. From the two possible paths that terminate at node 3 with input string *fine mead*, the determinization algorithm will pick one with the lowest accumulated weight, $\langle 7, JJ\_NN \rangle \oplus \langle 9, VB\_NN \rangle = \langle 7, JJ\_NN \rangle$, the expected result. Similarly, the determinization algorithm for the input *fine me* will result in picking the weight $\langle 5, VB\_PRP \rangle$. Thus, the determinization algorithm will produce the desired result for both input strings in Figure 4 and this can be shown to be true in general.

After determinization, the output symbols (tags) on the second weight may accumulate in certain paths, as in the example above. These weights need to be mapped back to associated input symbols (words). This mapping and the complete procedure for computing the single-best transduction paths for all unique input sequences for a given WFST (word lattice) using the $\langle \mathcal{T}, \mathcal{C} \rangle$ lexicographic semiring is described in the next few sections. Note that our categorial semiring allows for synchronizing the resulting output labels with their associated input labels, which the Povey et al. (2012) approach in general does not.

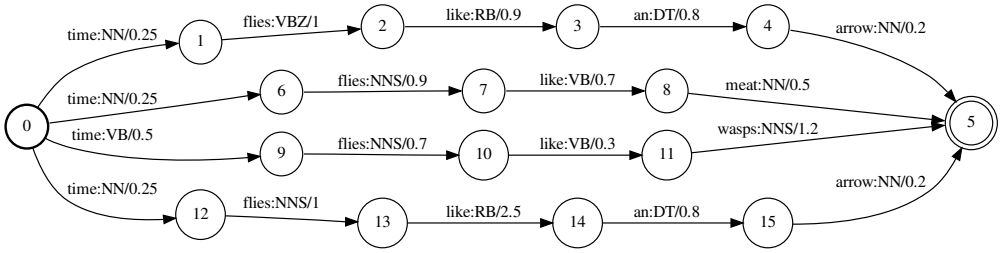
Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

3.3.1 Lattice Representation. Consider a lattice transducer where input labels are words (e.g., generated by a speech recognizer), output labels are tags (e.g., generated by a part-of-speech tagger), and weights in the tropical semiring represent negative log probabilities. For example, the toy lattice in Figure 5 has four paths, with two possible tag sequences for the string "*Time flies like an arrow*." In general for any given word sequence, there may be many paths in the lattice with that word sequence, with different costs corresponding to different ways of deriving that word sequence from the acoustic input, as well as different possible ways of tagging the input.

The procedure for removing all but the single best-scoring path for each input word sequence is as follows. We convert the weighted transducer to an equivalent acceptor in the $\langle \mathcal{T}, \mathcal{C} \rangle$-lexicographic semiring as in the algorithm in Figure 6. This acceptor is then determinized in the $\langle \mathcal{T}, \mathcal{C} \rangle$-lexicographic semiring, to yield a lattice where each distinct sequence of input-labels (words) corresponds to a single path. The result of converting the lattice in Figure 5 to the $\langle \mathcal{T}, \mathcal{C} \rangle$ semiring, followed by determinization, and conversion back to the tropical semiring, is shown in Figure 7. Note now that there are three paths, as desired, and that the tags on several of the paths are complex categorial tags.

We now have an acceptor in the $\langle \mathcal{T}, \mathcal{C} \rangle$-lexicographic semiring with, in general, complex categorial weights in the second component of the weight pair. It is now necessary to simplify these categorial weight sequences down to sequences of simplex categories, and reconstruct a

Figure 5
Sample input lattice.



Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics


```
CONVERT(L)
1  L′ ← new FST
2  for s in States(L) do
3      add state s′ to L′
4      if s is Start(L) then                          ▷ If (unique) start state
           Start(L′) ← s′
5      if Final(s, L) = ∞ then                        ▷ If state not final
           Final(s′, L′) ← ⟨∞, ∞_s⟩
6      else Final(s′, L′) ← ⟨Final(s, L), ϵ⟩
7      for arc in Arcs(s, L) do
8          ADDARC(L′, s′, Arc(next-state(arc)′, in-label(arc), in-label(arc),
                              ⟨weight(arc), out-label(arc)⟩))
9  return L′
```

Figure 6
Pseudocode for converting POS-tagged word lattice into an equivalent $\langle \mathcal{T}, \mathcal{C} \rangle$ lexicographic acceptor, with the arc labels corresponding to the input label of the original transducer.

transducer that maps words to tags with tropical weights. Figure 8 presents the result of such a simplification.

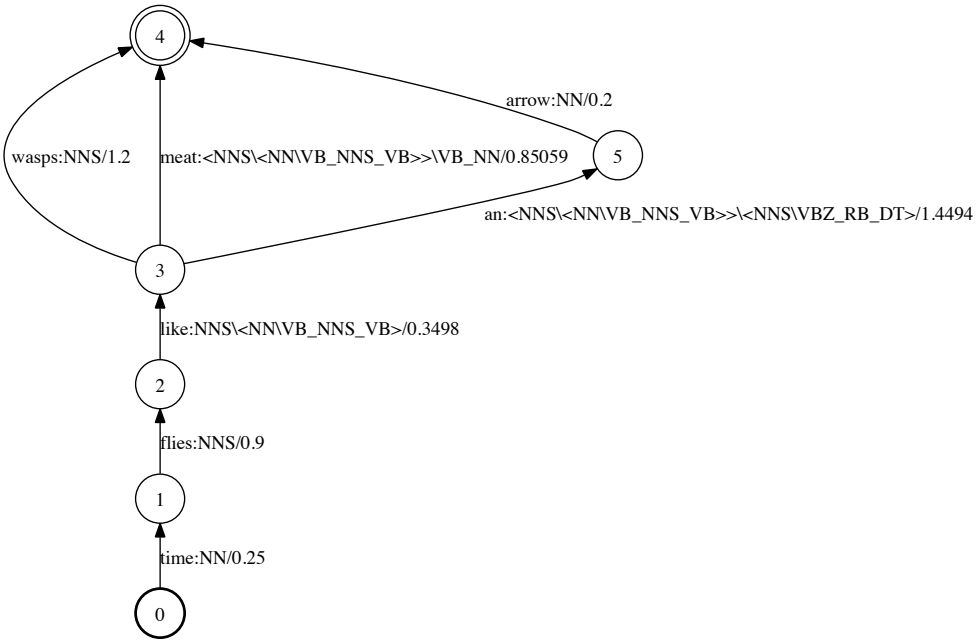
There are two approaches to this, outlined in the next two sections. The first involves pushing $\langle \mathcal{T}, \mathcal{C} \rangle$-lexicographic weights back from the final states, splitting states as needed, and then reconstructing the now simple categorial weights as output labels on the lattice. The latter reconstruction is essentially the inverse of the algorithm in Figure 6. The second approach involves creating a transducer in the tropical semiring with the input labels as words, and the output labels as complex tags. For this approach we need to construct a mapper transducer which, when composed with the lattice, will reconstruct the appropriate sequences of simplex tags.

3.3.2 State splitting and weight pushing. In the first approach we push weights back from the final states, thus requiring a reverse topological traversal of states in the lattice. The categorial weights of each arc are split into a prefix and a suffix, according to the SPLITWEIGHT function of Figure 9. The prefixes will be pushed towards the initial state, but if there are multiple prefixes associated with arcs leaving the state, then the state will need to be split: for k distinct prefixes, k distinct states are required. The PUSHSPLIT algorithm in Figure 10 first accumulates the set

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

of distinct prefixes at each state (lines 5-13), as well as storing the vector of arcs leaving the state, which will be subsequently modified. For each prefix, a new state is created (lines 22-25), although the first prefix in the set simply uses the state itself. Note that any categorial weight associated with the final cost yields the first prefix, meaning that it would be assigned the already existing state; hence all newly created states can be non-final. Each state is thus associated with a distinct single prefix, and each must be reachable from the same set of previous states as the original sate. Thus, for each new state, any arc that already has the original state as its destination state must be copied, and the new arc assigned the new destination state and weight, depending on the prefix associated with the new state (lines 26-30). The prefix associated with the original state must then be pushed onto the appropriate arcs (line 29). Finally, since all the prefix values have been pushed, each arc from the original state must be updated so that only the suffix value

Figure 7
Lattice after conversion to the $\langle \mathcal{T}, \mathcal{C} \rangle$ semiring, determinization, and conversion back to the tropical semiring.



Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

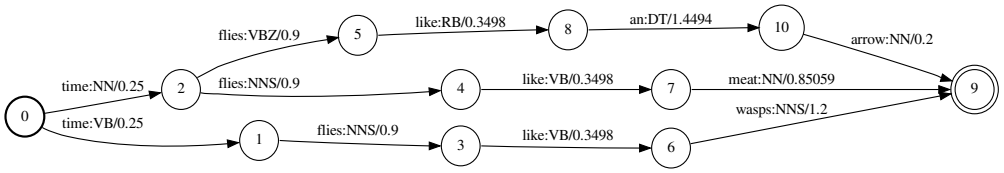


Figure 8
Final output lattice with the desired three paths.

remains in the weight, and now leaving the state associated with the original weight's prefix (lines 31-34).

Figure 9
Pseudocode for the SPLITWEIGHT algorithm on a categorial semiring. Returns a prefix, suffix pair for weight w.

SPLITWEIGHT(w)

```
1  if w = 0̄ or w = 1̄ then return w, w
2  if w is atomic then return 1̄, w
3  ▷ By construction, complex weights must end in an atomic weight, i.e., a simplex tag
4  if w = a\b, where b is atomic     ▷ either final atomic weight is preceded by a division
5     then return w, 1̄
6  let w = a b, where b is atomic    ▷ or final atomic weight is concatenated to the preceding
7  return a, b
```

3.3.3 Mapper approach. In the second approach, we build a *mapper* FST (M) that converts sequences of complex tags back to sequences of simple tags. The algorithm for constructing this mapper is given in Figure 11, and an illustration can be found in Figure 12. In essence, sequences of observed complex tags are interpreted and the resulting simplex tags are assigned to the output tape of the transducer. Simplex tags in the lattice are mapped to themselves in the mapper FST (line 6 of the function BUILDMAPPER in Figure 11), while complex tags require longer paths, the construction of which is detailed in the MAKEPATH function. The complex labels are parsed, and required input and output labels are placed on LIFO queues (lines 3-7). Then a path is created from state 0 in the mapper FST that eventually returns to state 0, labeled with the appropriate input and output sequences (lines 9-15).

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

PUSHSPLIT(L)

```
1  TopologicallySort(L)
2  for s in States(L) do          ▷ For each s, find all states with outgoing arcs with destination s
3      previous[s] ← COMPUTEPREVIOUSSTATES(s, L)
4  for s in Reverse(States(L)) do          ▷ Work on states in reverse topological order
5      prefixes ← ∅; arcs ← ∅              ▷ initialize prefix and arcs vectors
6      if FinalWeight(s) ≠ 0̄                ▷ If non-zero final weight, then:
7         then append Value2(FinalWeight(s)) to prefixes   ▷ Categorial component of final
8              Value2(FinalWeight(s)) ← 1̄   ▷   weight is a prefix; reset to 1̄
9      for a in Arcs(s, L) do               ▷ For all outgoing arcs from s
10         append a to arcs                 ▷ Store arc in arcs vector
11         prefix, suffix ← SPLITWEIGHT(Value2(Weight(a)))
12         if prefix not in prefixes        ▷ Store unique prefixes in vector
13            then append prefix to prefixes
14     DeleteArcs(s, L)                     ▷ Will replace with updated arcs later
15     previous-arcs ← ∅
16     for previous-s in previous[s] do      ▷ For all arcs with destination state s
17         for a in Arcs(previous-s, L) such that next-state(a) = s do
18             a′ ← a
19             delete a
20             append <previous-s, a′> to previous-arcs
21     new-states ← ∅
22     for prefix in prefixes do
23         if new-states = ∅          ▷ first prefix (from FinalWeight if non-zero) uses s
24            then new-states[prefix] ← s
25         else new-states[prefix] ← ADDNONFINALSTATE(L)
26         for <previous-s, arc> in previous-arcs do   ▷ For all arcs with destination s
27             a ← arc                                 ▷ create new arc
28             next-state(a) ← new-states[prefix]      ▷ update destination
29             Value2(Weight(a)) ← Value2(Weight(a)) ⊗ prefix   ▷ push prefix
30             ADDARC(L, previous-s, a)
31     for a in arcs do
32         prefix, suffix ← SPLITWEIGHT(Value2(Weight(a)))
33         Value2(Weight(a)) ← suffix   ▷ Categorial component of weight is now just suffix
34         ADDARC(L, new-states[prefix], a)   ▷ Origin state of arc is based on prefix
35 return
```

Figure 10
Pseudocode for the PUSHSPLIT algorithm on a lattice L in the $\langle \mathcal{T}, \mathcal{C} \rangle$ semiring. Note that Value2(w) for weight w is the categorial component of the weight. For the SPLITWEIGHT algorithm see Figure 9.

Once the mapper FST has been constructed, the determinized transducer is composed with the mapper — $L' \circ M$ — to yield the desired result, after projecting onto output labels. Note, crucially, that the mapper will in general change the topology of the determinized acceptor, splitting states as needed. This can be seen by comparing Figures 7 and 8. Indeed the mapping

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

approach and the PUSHSPLIT are completely equivalent, and, as we shall see, have similar time efficiency.

To understand the semantics of the categorial weights, consider the path that contains the words *flies like meat*, which has the categorial tag sequence

```
NNS       NNS\<NN\VB_NNS_VB>      <NNS\<NN\VB_NNS_VB>>\VB_NN
```

in Figure 7. The cancellation, working from right to left, first reduces

```
<NNS\<NN\VB_NNS_VB>>\VB_NN
```

with

```
NNS\<NN\VB_NNS_VB>
```

yielding

```
VB_NN
```

This then is concatenated with the initial simplex category to yield the sequence `NNS_VB_NN`. The actual cancellation is performed by the mapper transducer in Figure 12; the cancellation just described can be seen in the path that exits state 0, passes through state 3, and returns to state 0.

The construction in the case of the PUSHSPLIT algorithm is more direct since it operates on the determinized lattice *before it is converted back to the tropical semiring*; after which the simplex categories are reconstructed onto the output labels to yield a transducer identical to that in Figure 8.

3.4 Experimental Comparisons between Povey's and $\langle \mathcal{T}, \mathcal{C} \rangle$-lexicographic semirings

3.4.1 POS Tagging Problem. Our solutions were empirically evaluated on 4,664 lattices from the NIST English CTS RT Dev04 test set. The lattices were generated using a state-of-the-art speech recognizer, similar to Soltau et al. (2005), trained on about 2000 hours of data, that performed at a word error rate of about 24%. The utterances were decoded in three stages using speaker independent models, vocal-tract length normalized models and speaker-adapted models.

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

The three sets of models were similar in complexity with 8000 clustered pentaphone states and 150K Gaussians with diagonal covariances.

The lattices from the recognizer were tagged using a weighted finite state tagger. The tagger was trained on the Switchboard portion of the Penn Treebank (Marcus, Santorini, and Marcinkiewicz 1993). Treebank tokenization is different from the recognizer tokenization in some instances, such as for contractions ("don't" becomes "do n't") or possessives ("aaron's" becomes "aaron 's"). Further, many of the words in the recognizer vocabulary of 93k words are unobserved in tagger training, and are mapped to an OOV token "⟨unk⟩". Words in the treebank not in the recognizer vocabulary are also mapped to "⟨unk⟩", thus providing probability mass for that token in the tagger. A tokenization transducer $\mathcal{T}$ was created to map from recognizer vocabulary to tagger vocabulary.

Two POS-tagging models were trained: a first-order and a third-order hidden Markov model, estimated and encoded in tagging transducers $\mathcal{P}$. In the first-order HMM model, the transition probability is conditioned on the previous word's tag, while in the third-order model the transition probability is conditioned on the previous three word's tags. The transition probabilities are smoothed using Witten-Bell smoothing, and backoff smoothing is achieved using failure transitions. For each word in the tagger input vocabulary, only POS-tags observed with each word are allowed for that word, i.e., emission probability is not smoothed and is zero for unobserved tag/word pairs. For a given word lattice $\mathcal{L}$, it is first composed with the tokenizer $\mathcal{T}$, then with the POS tagger $\mathcal{P}$ to produce a transducer with original lattice word strings on the input side and tag strings on the output side.

These models were validated on a 2,000 sentence held-aside subset of the Switchboard treebank. The first-order model achieved 91.4% tagging accuracy, and the third-order model 93.8% accuracy, which is competitive for this particular task: Eidelman, Huang, and Harper (2010) reported accuracy of 92.4% for an HMM tagger on this task (though for a different

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

validation set). Both models likely suffer from using a single "⟨unk⟩" category, which is relatively coarse and does not capture informative suffix and prefix features that are common in such models for tagging OOVs. For the purposes of this paper, these models serve to demonstrate the utility of the new lexicographic semiring using realistic models. A similar WFST topology can be used for discriminatively trained models using richer feature sets, which would potentially achieve higher accuracy on the task.

The tagged lattices, obtained from composing ASR lattice with the POS tagger, were then converted to $\langle \mathcal{T}, \mathcal{C} \rangle$-lexicographic semiring, determinized in this lexicographic semiring and then converted back using the mapper transducer as discussed in Section 3.3.1. Note that the computational cost of this conversion is proportional to the number of arcs in the lattice and hence it is significantly less than the overhead incurred in the conventional approach of extracting all unique paths in the lattice and converting the paths back to a lattice after tagging.

The results of this operation were compared with the method of taking the 1,000 best paths through the original lattice, and removing any path where the path's word sequence had been seen in a lower-cost path. This generally resulted in a rank-ordered set of paths with $n < 1000$ members.

In all cases the n-best paths produced by the method proposed in this paper were identical to the n-best paths produced by the method just described. The only differences were due to minor floating-point number differences (expected due to weight-pushing in determinization), and cases where equivalent weighted paths were output in different orders.

3.4.2 Results. Despite large overall commonalities between Povey's approach, which we will refer to henceforth as "Povey", and $\langle T, C \rangle$ lexicographic approaches, which we refer to as "TC", there are some interesting differences between the two. One difference is that the highly structured categorial weights used in TC are more complex than the string weight used in Povey.

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

Another important difference in the approaches is the *synchronization* issue. In TC, the original input symbols are synchronized with determinized output symbols, whereas in Povey they are not. TC uses the semantics of the categorial grammar to keep the history of the operations during determinizing a lattice, whereas Povey lacks this semantics. While POS-tagging is a task that by definition has one tag per input token, many other tasks of interest – e.g., finding the most likely pronunciation or state sequence – will have a variable number of output labels per token, making synchronization in the absence of such semantics more difficult. Hence, these differences may affect time and space complexity, feasibility, and ease of use of the approaches in various tasks.

In this section, we compare the efficiency of the two approaches under the same situations on the same data. We ran the experiments detailed in section 3.4.1 in three conditions: Povey in the Kaldi toolkit (Povey et al. 2011) — with specialized determinization; and both approaches I and II in the OpenFST library (with general determinization). This allows us to tease apart the impact of the differences in the approaches that are due to the specialized determinization versus differences in the weight definitions. There would be nothing in principle to prevent the simultaneous epsilon removal being implemented in OpenFst for use with general determinization in $\langle T, C \rangle$ lexicographic semiring, though this is not the focus of this paper.

We compare these conditions in terms of running time, memory usage, and required disk space. Tables 1(a) and 1(b) show efficiency results of determinizing lattices tagged using the first-order HMM tagger, and tables 2(a), and 2(b) show those results for the third-order HMM tagger.

From Table 1(a) we see that Povey is faster and demands less memory compared to approach II. However, results using Povey with general determinization show that the memory demands between the two approaches are similar in the absence of the specialized determinization. We also see that the average number of intermediate tags produced during determinization in Povey is larger whereas the average length of intermediate tags is smaller than those in TC. This is

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

due to the fact that the categorial semiring keeps a complete history of operations by appending complex tags. We do not perform any special string compression on these tags, which may yield performance improvements (particularly with the larger POS-tagging model, as demonstrated in Table 2).

We compared the approach of using the mapper with that of the PUSHSPLIT algorithm in TC. The outputs were equivalent in both cases and the time and space complexities were comparable. The PUSHSPLIT algorithm was slightly more efficient than the mapper approach, however the difference is not significant.

While the intermediate space and processing time is larger for TC, we see from Table 1(b) that the output lattices resulting from TC are smaller than the output lattices in Povey in terms of number of states, transitions, input/output epsilons, and required disk space. Since the lattices produced by Povey are not synchronized, they contain many input/output epsilons, and

Table 1
For first-order HMM tagger, comparison of the two approaches for extracting the best and only the best POS for all the word sequences in the test lattie. The approach by Povey *et al* as implemented in Kaldi using a specialized determinization and our re-implementation in OpenFST with general determinization.

(a) Time, memory usage, disk space and intermediate tags (average per lattice).

	Povey (Povey et al. 2012)		TC	
			Push-Split	Mapper
Determinization	Special	General	General	General
running time (ms)	24.8	52.5	122.9	128.2
maximum resident set size (MB)	0.63	1.80	1.83	1.85
page reclaims	157	454	459	465
involuntary context switches	2	4	11	11
# of intermediate tags	49.1		26.6	
length of intermediate tags	1.8		2.4	

(b) Size of determinized lattices (average per lattice).

	Povey (Povey et al. 2012)		TC
Determinization	Special	General	General
# of states	344.7	768.2	187.5
# of transitions	565.5	1,295.6	433.5
# final states	7.6	7.6	7.7
# of input epsilons	190.1	602.6	16.7
# of output epsilons	91.7	299.2	0
# size of determinized lattice (KB)	15.1	31.4	11.6

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

therefore an increased number of states and transitions. In contrast, the lattices output by TC are synchronized and minimal. The size differences are even larger between the two approaches when both are using general determinization.

As tables 2(a), and 2(b) show, time and space efficiencies in tagging using the third-order HMM tagger follow the same pattern as those using the first-order HMM tagger, although the differences are more pronounced than in the former. We report these results on a subset of 4,000 out of 4,664 test lattices, chosen based on input lattice size so as to avoid cases of very high intermediate memory usage in general determinization. This high intermediate memory usage does argue for the specialized determinization, and was the rationale for that algorithm in Povey et al. (2012). The non-optimized string representation within the categorial semiring makes this even more of an issue for TC than Povey. Again, though, the size of the resulting lattice is much more compact when using the lexicographic $\langle T, C \rangle$ semiring. We leave investigation of an

Table 2
For third-order HMM tagger, comparison of the two approaches for extracting the best and only the best POS for all the word sequences in the test lattice. The approach by Povey *et al* as implemented in Kaldi using a specialized determinization and our re-implementation in OpenFST with general determinization.

(a) Time, memory usage, disk space and intermediate tags (average per lattice)

	Povey (Povey et al. 2012)		TC	
			Push-Split	Mapper
Determinization	Special	General	General	General
running time (sec)	2.9	9.6	49.9	50.7
maximum resident set size (MB)	28.0	62.8	240.9	241.1
page reclaims	7,890.9	16,589.9	62,963.7	63,002.8
involuntary context switches	270.1	895	4,985.4	4,915.2
# of intermediate tags	131.6		70.2	
length of intermediate tags	3.1		9.3	

(b) Size of determinized lattices (average per lattice)

	Povey (Povey et al. 2012)		TC
Determinization	Special	General	General
# of states	4,946.2	19,824.9	2,244.0
# of transitions	6,152.3	25,307.7	4,002.8
# final states	153.1	154.0	174.9
# of input epsilons	3,555.1	18,041.2	197.2
# of output epsilons	956.4	4,652.9	0
# size of determinized lattice (KB)	150.6	613.5	86.9

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

optimized string representation, such as storing the *history* only if it is different from the *value*, using the hash table data structure, or memory caching to future work.

4. Combining the semirings

In this paper, we have described two lexicographic semirings, each consisting of a weight pair. Suppose one wished to combine these two in a system that tags a lattice, then selects the single best tagging for each word sequence. An obvious way to do this would be to implement a two stage process. Apply the n-gram Markov model of the tagger with the backoff strategy implemented using the paired tropical semiring in the Section 2 with tags as acceptor labels. Then, convert the resulting transducer into the lexicographic $\langle T, C \rangle$ semiring with words as acceptor labels and determinize to obtain the correct results.

Since the lexicographic semiring is extensible, one might also think of combining the two semirings into a single $\langle T, T, C \rangle$ lexicographic *triple* where for example the first dimension is the failure arc cost, the second dimension holds the tag cost (n-gram transition costs of tags and the cost of observing the word given the tag), and the third dimension holds the tags represented in the categorial semiring. One might then compose the tagging model with the lattice, and then determinize in one step in the triple semiring.

While this works in the sense that it is technically possible to construct this semiring and determinize in it, it yields the wrong results. The reason for this is that the lexicographic semirings for the two tasks – the tagging task and the subsequent determinization of the tagged lattice – involve determinization with respect to different labels. In the first task, the backoff models are defined with respect to the Markov chain or n-grams of the tags and the labels on the resulting acceptor are tags. In the second task, the determinization needs to be performed with respect to the word labels to obtain unique tags for all word sequences. A cross product of the two types of labels would not accomplish the task either, since the determinzation would then produce unique paths for all word and tag combinations, and not the best tag sequences for all

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

word sequences. There is no obvious or easy way to determinize with respect to both sets of labels simultaneously.

We can illustrate this problem with an example, which is also useful for clearly understanding how each of the semirings functions. The simple example involves a cost-free word lattice consisting of two paths $a\ a$ and $b\ a$, in a scenario where word a can take two possible tags A or B. We will assign variables to model costs, so that we can illustrate the range of scenarios where the use of the triple semiring will yield an incorrrect answer, and why. Let $c(a{:}A)$ be the cost of the tag A with word a, which in our HMM POS tagger is -log P($a \mid A$). Let $g(x, y)$ be the cost in the grammar (tag sequence model) of transitioning from state x to state y in the model. See Figure 13 for our example L, T, $L \circ T$ and G. All costs in the example are in the $\langle \mathcal{T}, \mathcal{T} \rangle$ semiring for ease of explication; the first dimension of the costs is zero except for backoff arcs in G.

In Figure 14 we show the result of $L \circ T \circ G$ both after simple composition and after epsilon removal and conversion from a transducer in the $\langle \mathcal{T}, \mathcal{T} \rangle$ semiring to an acceptor in the $\langle \langle \mathcal{T}, \mathcal{T} \rangle, \mathcal{C} \rangle$ semiring. In the second and third WFSTs, we highlight the paths that have zero cost in the first dimension of that semiring, which are the only paths that can result from determinization (whatever the model costs). These paths only include tag B for the initial instance of symbol a. However, if $g(0,2) + c(a{:}A) + g(2,3) + c(a{:}A) + g(3,3) < c(a{:}B) + g(0,1) + c(a{:}A) + g(1,3)$ then the tag sequence a:A a:A would have lower (second dimension) cost than a:B a:A, despite having taken a backoff arc. Since using a backoff arc is the only way to produce the tag sequence $A\ A$, then that path should be the result. In order to get the correct result, one must first determinize with x:Y labels as unit (using fstencode) in the $\langle \mathcal{T}, \mathcal{T} \rangle$ semiring; then project into the $\langle \mathcal{T}, \mathcal{C} \rangle$ semiring and determinize again.

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

5. Conclusions

In this paper we have introduced two applications of lexicographic semirings to speech and language processing problems. The first application used the $\langle \mathcal{T}, \mathcal{T} \rangle$ lexicographic semiring to provide an exact encoding of failure arcs in an n-gram language model using an epsilon representation for the failure arc. This *lexicographic language model semiring* allows much more flexibility in combining the language model with other linguistic models without danger of prohibitive blow-up in the size of the resulting transducers: for example, precomposing the language model with a lexicon and a context model in a CLG model of speech recognition (Mohri, Pereira, and Riley 2002).

The second application was of a $\langle \mathcal{T}, \mathcal{C} \rangle$ lexicographic semiring to the problem of determinizing a tagged word lattice so that each word sequence has the single best tag sequence. This was accomplished by encoding the tags as the second dimension of the $\langle \mathcal{T}, \mathcal{C} \rangle$ semiring, then determinizing the resulting acceptor. Finally we map the second dimension categorial weights back as output labels. This latter stage generally requires that we push complex categorial weights back to reconstruct a sequence of simplex categories, an operation that can be performed in two distinct and equally efficient ways. As part of this work we developed a novel string semiring, the categorial semiring, which we have described in detail for the first time here.

For both of these applications, the lexicographic semiring solution was shown to be competitive in terms of efficiency with alternative approaches.

In the future, one can imagine various extensions of the core ideas presented here to further applications. For example one might use an Optimality-Theory-inspired model with ranked constraints implemented using a lexicographic semiring as part of a pronunciation modeling system that ranks pronunciations according to the degree to which they violate various constraints of the language.. The $\langle \mathcal{T}, \mathcal{C} \rangle$-lexicographic semiring introduced in Section 3 can be generalized to compute the single-best transduction path in multi-tape weighted transducers. For

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

instance, by encoding the arc likelihoods, the phone sequence, the clustered allophone sequence, acoustic state sequence, and acoustic segmental duration associated with word sequence as a $\langle \mathcal{T}, \mathcal{C}, \mathcal{C}, \mathcal{C}, \mathcal{T} \rangle$ lexicographic semiring and determinizing the resulting automaton, we can extract the tags corresponding to the single-best word sequence. Thus, our method is much more flexible and powerful than algorithms developed specifically for determinizing POS-tagged word lattices as in Roche and Schabes (1995) or approximations specific to applications as in Shugrina (2010).

All of the software described in this paper is publicly available. The lexicographic semiring is distributed as part of the core OpenFst distribution at `http://www.openfst.org`. The categorial semiring is available in the contributed section at `http://www.openfst.org/twiki/bin/view/Contrib/FstContrib`. The categorial rescoring methods including both the mapping and push-split approaches are available from `http://www.opengrm.org`.

References

Abney, Steven. 1996. Partial parsing via finite-state cascades. *Natural Language Engineering*, 2(4):337–344.

Albro, Daniel. 1998. Three formal extensions to Primitive Optimality Theory. In *COLING-1998*, pages 21–25, Montreal.

Allauzen, Cyril, Mehryar Mohri, and Brian Roark. 2003. Generalized algorithms for constructing statistical language models. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 40–47.

Allauzen, Cyril, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. 2007. OpenFst: A general and efficient weighted finite-state transducer library. In *Proceedings of the Twelfth International Conference on Implementation and Application of Automata (CIAA 2007), Lecture Notes in Computer Science*, volume 4793, pages 11–23, Prague, Czech Republic. Springer.

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

Bangalore, Srinivas and Giuseppe Riccardi. 2001. A finite-state approach to machine translation. In *Second Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 1–8.

de Gispert, Adrià, Gonzalo Iglesias, Graeme Blackwood, Eduardo Banga, and William Byrne. 2010. Hierarchical phrase-based translation with weighted finite-state transducers and shallow-*n* grammars. *Computational Linguistics*, 36(3):505–533.

Eidelman, Vladimir, Zhongqiang Huang, and Mary Harper. 2010. Lessons learned in part-of-speech tagging of conversational speech. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 821–831.

Eisner, Jason. 1998. FOOTFORM decomposed: Using primitive constraints in OT. In *Proceedings of SCIL VIII*, volume 31, pages 115–143. MIT Working Papers in Linguistics.

Eisner, Jason. 2000. Directional constraint evaluation in Optimality Theory. In *COLING-2000*, pages 257–263, Saarbrücken.

Eisner, Jason. 2001. Expectation semirings: Flexible EM for learning finite-state transducers. In *Proc. of the ESSLLI Workshop on Finite-State Methods in NLP (FSMNLP)*, pages 1–5, Helsinki.

Ellison, T. Mark. 1994. Phonological derivation in Optimality Theory. In *COLING-1994*, pages 1007–1013, Kyoto.

Frank, Robert and Giorgio Satta. 1998. Optimality theory and the generative complexity of constraint violability. *Computational Linguistics*, 24:307–315.

Golan, Jonathan. 1999. *Semirings and their Applications*. Kluwer Academic Publishers, Dordrecht.

Joshi, Aravind and Phil Hopely. 1996. A parser from antiquity. *Natural Language Engineering*, 2(4):291–294.

Karttunen, Lauri. 1998. The proper treatment of optimality in computational phonology. In *Proceedings of the International Workshop on Finite-State Methods in Natural Language Processing*, pages 1–12, Bilkent University, Ankara.

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

Koskenniemi, Kimmo. 1983. *Two-Level Morphology: A General Computational Model of Word-Form Recognition and Production*. Ph.D. thesis, University of Helsinki.

Kuich, Werner and Arto Salomaa. 1986. *Semirings, Automata, Languages*. Number 5 in EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, Germany.

Lambek, Joachim. 1958. The mathematics of sentence structure. *American Mathematical Monthly*, 65(3):154–170.

Lehr, Maider and Izhak Shafran. 2011. Learning a discriminative weighted finite-state transducer for speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(5):1360–1367, July.

Marcus, Mitch, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Mohri, Mehryar. 2002. Semiring framework and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350.

Mohri, Mehryar. 2009. Weighted automata algorithms. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, Monographs in Theoretical Computer Science. Springer, pages 213–254.

Mohri, Mehryar, Fernando C. N. Pereira, and Michael Riley. 2002. Weighted finite-state transducers in speech recognition. *Computer Speech and Language*, 16(1):69–88.

Povey, Daniel, Arnab Ghoshal, Gilles Boulianne, Lukáš Burget, Ondřej Glembek, Nagendra Goel, Mirko Hanneman, Petr Motlíček, Yanmin Qian, Petr Schwarz, Jan Silovský, Georg Stemmer, and Karel Veselý. 2011. The Kaldi speech recognition toolkit. In *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*.

Povey, Daniel, Mirko Hannemann, Gilles Boulianne, Lukáš Burget, Arnab Ghoshal, Milos Janda, Martin Karafiat, Stefan Kombrink, Petr Motlíček, Yanmin Qian, Korbinian Riedhammer, Karel Veselý, and

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

Ngoc Thang Vu. 2012. Generating exact lattices in the WFST framework. In *IEEE International Conference on Acoustic, Speech, and Signal Processing (ICASSP)*, pages 4213–4216.

Prince, Alan and Paul Smolensky. 2004. *Optimality Theory: Constraint Interaction in Generative Grammar*. Blackwell, Oxford.

Roark, Brian, Murat Saraclar, and Michael Collins. 2007. Discriminative n-gram language modeling. *Computer Speech and Language*, 21(2):373–392.

Roark, Brian and Richard Sproat. 2007. *Computational Approaches to Morphology and Syntax*. Oxford University Press, Oxford.

Roark, Brian, Richard Sproat, Cyril Allauzen, Michael Riley, Jeffrey Sorensen, and Terry Tai. 2012. The OpenGrm open-source finite-state grammar software libraries. In *Proceedings of the Association for Computational Linguistics, System Demonstrations*, pages 61–66, Jeju Island, South Korea. Association for Computational Linguistics.

Roark, Brian, Richard Sproat, and Izhak Shafran. 2011. Lexicographic semirings for exact automata encoding of sequence models. In *Proceedings of ACL-HLT, 2011*, volume 2, pages 1–5, Portland, OR. Association for Computational Linguistics.

Roche, Emmanuel and Yves Schabes. 1995. Deterministic part-of-speech tagging with finite-state transducers. *Computational Linguistics*, 21:227–253, June.

Shafran, Izhak, Richard Sproat, Mahsa Yarmohammadi, and Brian Roark. 2011. Efficient determinization of tagged word lattices using categorial and lexicographic semirings. In *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 283–288.

Shugrina, Maria. 2010. Formatting time-aligned asr transcripts for readability. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 198–206, Stroudsburg, PA, USA. Association for Computational Linguistics.

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

Soltau, Hagen, Brian Kingsbury, Lidia Mangu, Daniel Povey, George Saon, and Geoffrey Zweig. 2005. The IBM 2004 conversational telephony system for rich transcription. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 205–208.

Sproat, Richard. 1996. Multilingual text analysis for text-to-speech synthesis. *Natural Language Engineering*, 2(4):369–380.

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics


```
BUILDMAPPER(L)
 1  for s in States(L), arc in Arcs(s, L) do
 2      SYMBOLS ← output-label(arc)            ▷ set of symbols labeling lattice arcs
 3  M ← new FST
 4  Start(M) ← 0; Final(M) ← 0                 ▷ single state is both start and final state
 5  for λ in SYMBOLS do
 6      if ISSIMPLE(λ) then ADDARC(M, 0, Arc(0, λ, λ, 1̄)) else MAKEPATH(M, λ)
 7  return M

MAKEPATH(M, λ)
 1  outputs ← EXTRACTTRAILINGSYMBOLS(λ)        ▷ LIFO input queue
 2  inputs ← ∅                                 ▷ LIFO output queue
 3  ENQUEUE(λ, inputs)
 4  while λ ≠ 1̄
 5      λ′, ν ← PARSELABEL(λ)
 6      if λ′ ≠ λ then ENQUEUE(λ′, inputs)
 7      λ ← ν
 8  s ← 0
 9  while |inputs| > 0 or |outputs| > 0        ▷ create path from state 0 with inputs/outputs
10      a ← Arc(0, ϵ, ϵ, 1̄)                    ▷ default ϵ arc with destination state 0
11      if |inputs| > 0 then input-label(a) ← DEQUEUE(inputs)
12      if |outputs| > 0 then output-label(a) ← DEQUEUE(outputs)
13      if |inputs| > 0 or |outputs| > 0 then next-state(a) ← ADDNONFINALSTATE(M)
14      ADDARC(M, s, a)
15      s ← next-state(a)
16  return

EXTRACTTRAILINGSYMBOLS(λ)
 1  outputs ← ∅                                ▷ LIFO output queue
 2  λ̄ ← MAKESYMBOLQUEUE(λ)                     ▷ LIFO queue of symbols in string order, incl. delimiters
 3  while |λ̄| > 0
 4      a ← DEQUEUE(λ̄)
 5      if a = backslash then return outputs
 6      if a ≠ left-bracket and a ≠ right-bracket then ENQUEUE(a, outputs)
 7  return outputs

PARSELABEL(λ)
 1  λ ← STRIPOUTERBRACKETS(λ)
 2  if λ contains no backslash then return λ, 1̄
 3  Let λ = δ\ν for rightmost (non-embedded) backslash   ▷ denominator \ numerator
 4  return STRIPOUTERBRACKETS(δ), STRIPOUTERBRACKETS(ν)
```

Figure 11
Pseudocode for construction of mapper transducer. The function ISSIMPLE returns true in case the tag λ is a simple tag, not a complex categorial tag.

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics

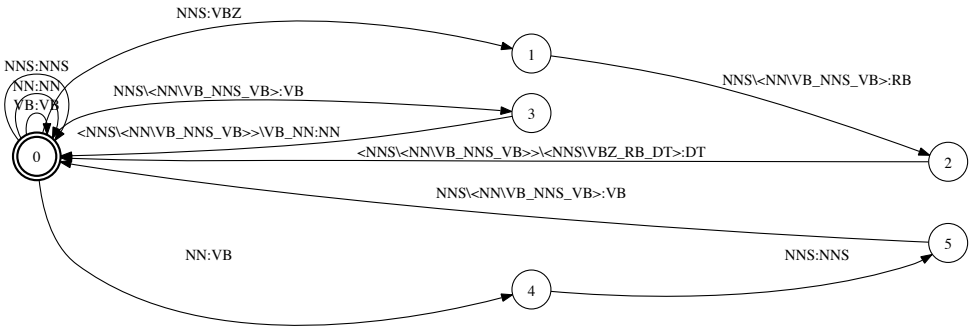


Figure 12
After conversion of the $\langle \mathcal{T}, \mathcal{C} \rangle$ lattice back to the tropical, this mapper will convert the lattice to its final form.

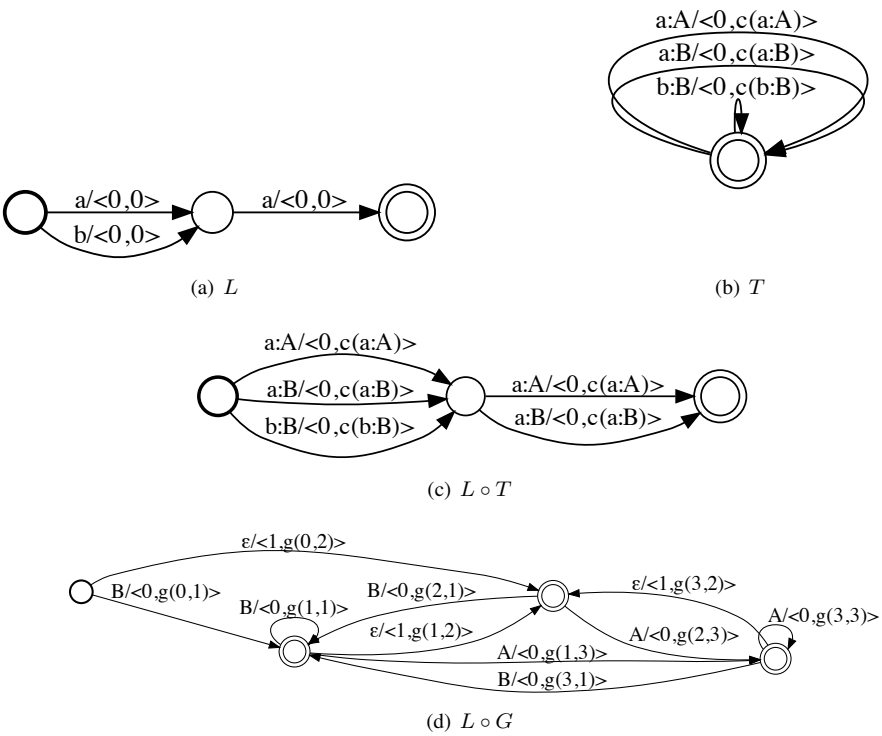
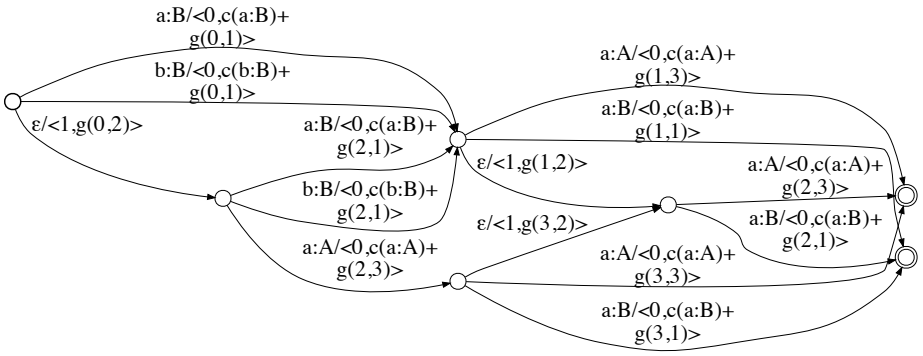
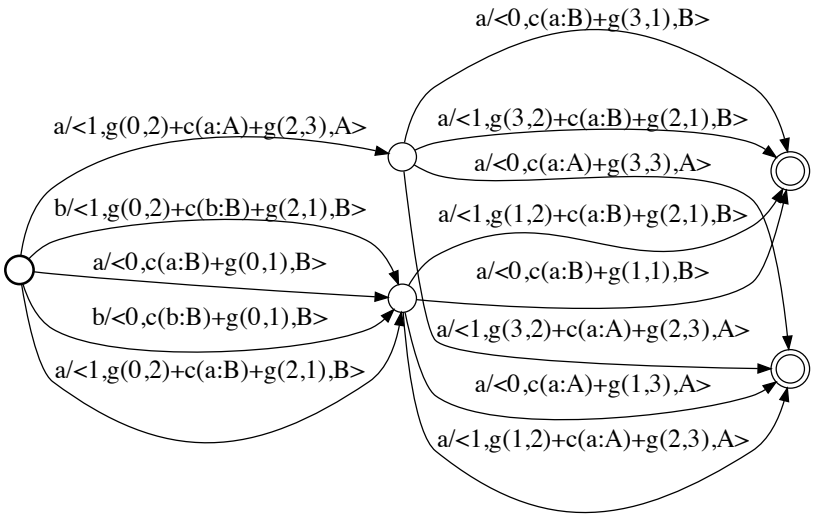


Figure 13
Input unweighted lattice L and tag mapper transducer T in $\langle \mathcal{T}, \mathcal{T} \rangle$ semiring, where $c(x{:}Y)$ is the cost of word x with tag Y. When composed, $L \circ T$ yields a lattice of word:tag sequences. G is a tag language model, which encodes the smoothed transition probabilities of the HMM tagger. ϵ represents backoff transitions; and $g(x, y)$ gives the cost of transitioning from state x to state y in the model. Again, costs are in the $\langle \mathcal{T}, \mathcal{T} \rangle$ semiring, so that backoff transitions have a cost of 1 in the first dimension.

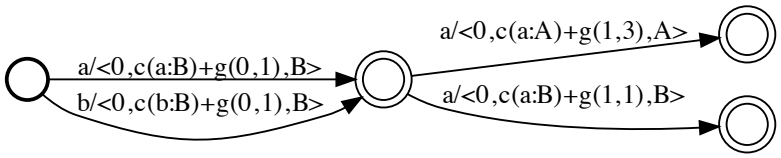
Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics



(a) $L \circ T \circ G$



(b) $L \circ T \circ G$ after epsilon removal and conversion to triple semiring



(c) Paths with 0 cost in first dimension (only possible resulting paths after determinization)

Figure 14
Full FST after composing $L \circ T \circ G$ and then following epsilon removal and conversion to $\langle\langle \mathcal{T}, \mathcal{T} \rangle, \mathcal{C}\rangle$ "triple" semiring. Only four paths have zero cost (i.e., no backoff arcs taken) through the resulting automaton, and these are the only possible paths after determinization.

Computational Linguistics Just Accepted MS.
doi: 10.1162/COLI_a_00198
© Association for Computational Linguistics